View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Research Online

University of Wollongong

Research Online

Faculty of Engineering and Information Sciences - Papers: Part A

Faculty of Engineering and Information Sciences

2001

MAT: a mobile agent system for supporting autonomous mobile agents

Wei Li
Central Queensland University

Minjie Zhang
University of Wollongong, minjie@uow.edu.au

Follow this and additional works at: https://ro.uow.edu.au/eispapers

 Part of the Engineering Commons, and the Science and Technology Studies Commons

Recommended Citation

Li, Wei and Zhang, Minjie, "MAT: a mobile agent system for supporting autonomous mobile agents" (2001). *Faculty of Engineering and Information Sciences - Papers: Part A*. 2648.
https://ro.uow.edu.au/eispapers/2648

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

MAT: a mobile agent system for supporting autonomous mobile agents

Abstract

Mobile Agent Template (MAT) is a mobile agent system that is under study and development at the Institute of Computing Technology, Chinese Academy of Sciences and sponsored by the University of Wollongong, Australia. MAT is not an alternative to other mobile agent systems, but is an agent system that can provide the autonomy to mobile agents. MAT tries to support new Web applications, such as the mobile computation, by autonomous and mobile agents. Mobile Thread Programming Model (MTPM), Distributed Task Plan (DTP) and Active State Space (ASS) are integral components on which MAT is constructed. Integration of these three components provides agents with an autonomous work mode and an autonomy- supporting execution environment. In this paper, we define autonomies of agents in the context of mobility and propose our autonomous theories, which are autonomous workflow, asynchronous and localised interactions, and a virtual supporting environment. This paper also outlines current implementation mechanisms of MAT including architecture, program paradigm, distributed task planning and communications. The main contributions of this research are that: (1) workflows are adopted as agents' working modes; (2) a goal-directed and dynamic task planning is used to deal with the heterogeneity and dynamism of networks; and (3) a virtually platform-independent environment is constructed to provide mobile agents with asynchronous, anonymous and fully localised interactions. The innovation of this research is to provide a new solution for novel Web applications such as mobile computations by using MAT.

Keywords

supporting, agents, mat, agent, mobile, autonomous, system

Disciplines

Engineering | Science and Technology Studies

Publication Details

Li, W. & Zhang, M. (2001). MAT: a mobile agent system for supporting autonomous mobile agents. Journal of Research and Practice in Information Technology, 33 (3), 210-226.

This journal article is available at Research Online: https://ro.uow.edu.au/eispapers/2648

MAT: a Mobile Agent System for Supporting Autonomous Mobile Agents

Wei Li

Faculty of Informatics and Communication
Central Queensland University
North Rockhampton, QLD 4702, Australia
w.li@cqu.edu.au

Minjie Zhang

School of Information Technology and Computer Science
University of Wollongong
NSW 2522, Australia
minjie@uow.edu.au

Mobile Agent Template (MAT) is a mobile agent system that is under study and development at the Institute of Computing Technology, Chinese Academy of Sciences and sponsored by the University of Wollongong, Australia. MAT is not an alternative to other mobile agent systems, but is an agent system that can provide the autonomy to mobile agents. MAT tries to support new Web applications, such as the mobile computation, by autonomous and mobile agents. Mobile Thread Programming Model (MTPM), Distributed Task Plan (DTP) and Active State Space (ASS) are integral components on which MAT is constructed. Integration of these three components provides agents with an autonomous work mode and an autonomy-supporting execution environment. In this paper, we define autonomies of agents in the context of mobility and propose our autonomous theories, which are autonomous workflow, asynchronous and localised interactions, and a virtual supporting environment. This paper also outlines current implementation mechanisms of MAT including architecture, program paradigm, distributed task planning and communications. The main contributions of this research are that: (1) workflows are adopted as agents' working modes; (2) a goal-directed and dynamic task planning is used to deal with the heterogeneity and dynamism of networks; and (3) a virtually platform-independent environment is constructed to provide mobile agents with asynchronous, anonymous and fully localised interactions. The innovation of this research is to provide a new solution for novel Web applications such as mobile computations by using MAT.

Keywords: autonomous agent, mobile agent system, distributed supporting environment, thread migration

Copyright© 2001, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: March 2000, revised May 2001
Associate Editor: Hugh Williams

1 INTRODUCTION

It is believed that a new star in the sky of distributed object-oriented systems is the emerging field of mobile agents (Straßer, Baumann and Hohl, 1997). There are many mobile agent systems (Magic, 1998; Gray, 1996; Lange and Oshima, 1998; Object Space) that have been developed since the first prototype, Telescript (White, 1997), was proposed for e-commerce in 1994. There are many ongoing discussions about developments of new mobile agent systems, but few have focused on both mobility and autonomy (Bic, Fukuda and Dillencount, 1996; Cai, Gloor and Nog, 1996). In an application of mobile computation (Gray, *et al*, 1997; Kotz, *et al*, 1997), a user often launches a mobile agent from a laptop that is connected to the Internet, then the user disconnects the laptop from the Internet. The mobile agent travels in the Internet autonomously, retrieving and updating information locally on behalf of its owner. Later, the mobile agent will return to the user's laptop and report the results when the user's laptop is reconnected to the Internet. Mobile agents should have the "intelligence" to perform self-contained navigation and computation, giving mobile agents the powers to adapt to dynamic and heterogeneous networks. This is necessary because in most cases mobile agents cannot interact with their owners. However, it is not easy to define what autonomy is and how to provide mobile agents with autonomies.

This paper focuses on providing agents with autonomies in the context of mobility. In this paper, new concepts, paradigms, and models are proposed and studied as the bases of Mobile Agent Template (MAT) and an agent structure, task description and hosting environment are integral parts to support the autonomies of mobile agents. The contributions of this research are that we adopt the workflow as agents' working mode; we use goal-directed and dynamic task planning to deal with the heterogeneity and dynamism of networks; and we construct a virtually platform-independent environment to provide mobile agents with asynchronous, anonymous and fully localised interactions.

This paper is organised as follows: In Section 2, three kinds of autonomies are defined for clarifying the meaning of autonomy in the context of mobility. In Section 3, the autonomous methodologies, which are Distributed Task Plan (DTP), Active State Space (ASS) and mechanisms for fully localised interactions, are proposed for designing the architecture of MAT. In Section 4, the implementation mechanisms of MAT, including autonomous primitives, DTP planning, and reflections of ASS are addressed. Finally in Section 5, this paper is concluded and directions of future research are given.

2 WHAT ARE AUTONOMIES OF MOBILE AGENTS?

Autonomy is a difficult concept to pin down precisely, but "it means that the system should be able to act without the direct intervention of humans (or other agents), and that it should have control over its own actions and internal state" (Jennings, Sycara, and Wooldridge, 1998). In the context of mobile agents, it is even more difficult to define autonomy. Having investigated some Web applications such as Internet information retrieval and mobile computation for which mobile agents are suitable (Ciancarini, Tolksdorf and Vitali, 1998; Rus, Gray and Kotz, 1997), we identify the necessary features that an autonomous mobile agent should have. By autonomy, a mobile agent is self-contained in navigation, computation and communication when transporting through the underlying heterogeneous and dynamic networks. An autonomous mobile agent should be aware of where to go to find messages or resources, what actions to perform for getting information, and how to deal with exceptions when its performances fail. In the context of autonomy, it should be enough for mobile agents to interact with communication partners only in an asynchronous, anonymous and

localised mode for inter-agent cooperation or resource access. Based on the above features of mobile agents, we have defined the following three types of autonomies in the following subsection.

2.1 Definitions of autonomies

Definition 1: *Mobile autonomy* is the capability of self-navigation of mobile agents through the underlying networks.

The mobile autonomy of mobile agents should be inherent. Mobile agents can be capable of making decisions about their destinations, navigation modes and where to dock when a destination is not reachable. The navigation modes could be serial or in parallel. Normally, a mobile agent completes a distributed task by visiting a series of network nodes serially and processing queries and updating information locally. However, a mobile agent can clone itself and assign a subtask to each of it's duplicates for executing a distributed task concurrently when multiple resources are available and a distributed task can be divided into concurrently executing subtasks. The solutions of subtasks from different mobile agents are combined into a result finally. A mobile agent should also know where to dock temporarily in order to avoid getting lost when a network connection is broken or unavailable, such as the disconnection of a laptop computer from the network. So autonomous mobile agents are dynamically network-aware entities on which Web applications are constructed.

Definition 2: *Computational autonomy* is that a mobile agent can get enough computational functions for accomplishing a distributed task.

A mobile agent should make use of all kinds of computational resources. A mobile agent can invoke functions in different processes or load functions into its process for execution when those functions reside at its visiting network nodes. A mobile agent can also complete a special computational task by executing its own functions carried by the agent when visiting a network node with desirable resources but without necessary computational functions.

Definition 3: *Communicational autonomy* is that a mobile agent can send and receive messages in an asynchronous, anonymous and indirect way.

2.2 Discussion and classification of interactions

However, no matter what degree of autonomy they have, mobile agents need *inter-agent* interactions for cooperation and *agent-hosting execution environment* interactions for resource accesses because any autonomous mobile agent exists in a community, which may be composed of other agents and heterogeneous hosting environments. Besides the interactions with their owners at their home machines, autonomous mobile agents need at least two kinds of interactions during their lifecycles.

1. *Inter-agent interactions* are needed when a Web application consists of more than one mobile agent. Let's consider the following scenario. When a mobile agent realises that its task can be concurrently executed, for example, a mobile agent finds other n interested links in a HTML page while retrieving contents of the page, the mobile agent will duplicate $n-1$ instances with respective retrieval tasks. The agent and its $n-1$ duplicates will transport to n interesting sites for further information retrievals. *Inter-agent* interactions occur when results from different agents are synthesised into a final result for submission to the agent's owner at the home machine.
2. *Agent-hosting execution environment interactions* are needed when a mobile agent accesses resources and services in a local execution environment of network nodes it visits. All the information in a hosting execution environment must be structured for mobile agent accessing, and protocols for retrieving or updating information must be carefully defined.

There are new features in interactions among autonomous mobile agents or between autonomous mobile agents and hosting execution environments. In the context of autonomy, interactions should be fully asynchronous, anonymous and localised between communication partners. Firstly, the asynchronism does not require a strict agreement on communication time while the synchronism implies that any interaction between communication partners depends on complex protocols and synchronous mechanisms. The code load required for mobile agents to deal with their protocol, and synchronous mechanism is too high. Mobile agents would have too many code loads for the protocol and the synchronous mechanism accomplishments. Secondly, if an autonomous mobile agent must know the communication partners' name for interaction, naming services would be involved in agent communications. The naming services result in extra network (not local) interactions between mobile agents and naming service agencies. Too many interactions might weaken the autonomy we pursue in many Web applications. Also, it is not necessary for an autonomous mobile agent to know other agents' names because it is enough for an agent to interact with others as long as it can identify that its communication partners and itself belong to the same application. Finally, any interaction should be localised. An important advantage of mobile agents over other distributed technologies is the locality of distributed information processing. Locality leads to advantages in terms of high efficiency, reliability and flexibility. So interactions between autonomous mobile agents and their communication partners should not introduce any extra network communication.

In the concept of wireless networks (low band) and mobile computation (partially connected devices), minimisation of network communications and localisation of interactions are the desirable goals of MAT. To achieve those aims, we cannot use the traditional communication models, such as the direct communication model and the rendezvous model (Acharya, Ranganathan and Saltz, 1996; Baumann, Hohl, and Radouniklis, 1997; Gray 1996), or the meeting-oriented model (Magic, 1998; Peine and Stolpmann, 1997). Instead, we design a new model: *ASS (Active State Space)*, which is a third party to facilitate interactions between autonomous mobile agents and their communication partners for cooperation or resource accesses.

3 HOW TO SUPPORT AUTONOMIES: METHODOLOGY OF MAT

In this section, we demonstrate how to introduce desirable autonomies into mobile agents by using the virtual autonomy-supporting environment and the autonomy-supporting work mode.

3.1 Abstract description to heterogeneous execution environments in MAT

Mobile agents exist in heterogeneous networks such as the Internet. Resources are formatted and managed with different modes. Accessing different system resources directly by mobile agents results in the following limitations:

1. Increasing complexity of the application-level programming. It is evident that programmers must write different code for accessing different type or the same type of resources. For example, an *html* file is described by path */user/wli/document/index.html* in UNIX and described by path *c:\wli\document\index.html* in Windows. In order to access resources in heterogeneous networks, programmers should be familiar with different networks and operating systems.
2. Increasing code loads carried by mobile agents. The more heterogeneous the network that a mobile agent travels in, the heavier the code load that a mobile agent carries. A mobile agent with a too heavy code load is not a compact entity suitable for Web applications in the context of minimising network communications to deal with low band and unreliable networks such as the wireless network.

3. Decreasing reusability of agent codes. Any effort to extend a mobile agent-based application will result in the modification and addition of agent code.

In MAT, we use Active State Space (ASS) to provide an integrated description of heterogeneous resources that are accessed by mobile agents. ASS abstracts resources of different systems into data integrity so as to deal with the heterogeneity and the dynamism of the hosting execution environments. Mobile agents use virtual resources to access the real resources. ASS is responsible for mapping between abstract resources and real resources. ASS provides mobile agents with platform-independent resource accesses. In this point, ASS is quite similar to JVM, which provides platform-independent Java instructions to programmers. Another motivation for use of virtual resources is to provide a secure interface between mobile agents and system resources. It is necessary for ASS to perform security checks on some sensitive operations such as deleting files because a malicious or badly programmed agent may possibly contain dangerous code. Comparison of direct resource access to virtual resource access is graphically illustrated in Figure 1.

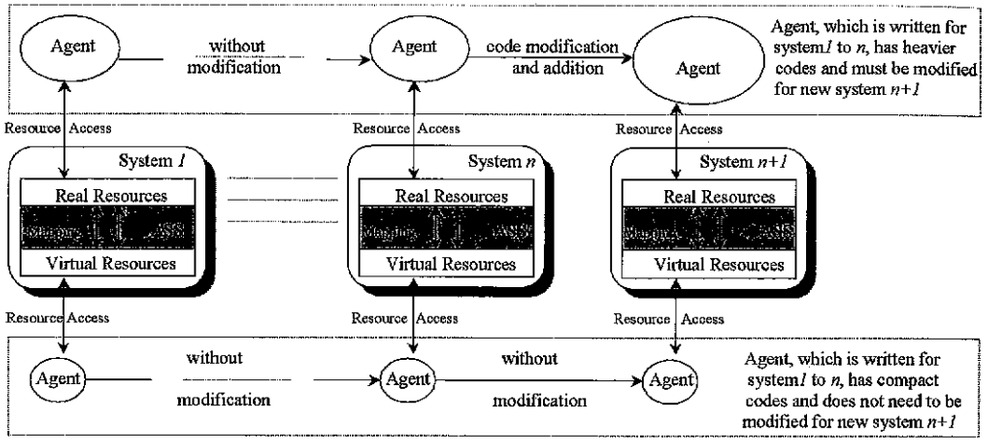

Figure 1: Comparison of direct resource access to virtual resource access

3.2 Asynchronous and localised interactions between communication partners in MAT

Asynchronous, anonymous and localised interactions are main features that must be satisfied in the context of Web applications based on autonomous mobile agents. In MAT, we use Resource Space, a component of ASS, to coordinate actions of mobile agents for cooperation in a task accomplishment. We use the following scenario to demonstrate the asynchronous, anonymous and localised coordination model for *inter-agent* interactions.

When a mobile agent *PA* knows that its distributed task DT can be executed concurrently on destinations D_1, D_2, ..., D_{n-1}, and D_n, the agent PA (called Parent) duplicates *n-1* instances SA_1, SA_2, ..., and SA_{n-1} with respective subtasks ST_1, ST_2, ..., and ST_{n-1}. Each duplicate will transport to D_1, D_2, ..., and D_{n-1} respectively, and finally, the parent agent *PA* will transport to the destination D_n with subtask ST_n. Subtasks ST_1, ST_2, ..., ST_{n-1} and ST_n can be the same or different. If our aim is to exploit distributed resources for executing distributed tasks for fast response, the subtasks should be same, such as, getting the same program table from different TV web sites. If our aim is to exploit distributed resources for retrieving different information, the subtasks should be different, such as, getting different price table of the same travel from different travel agencies. Because SA_1, SA_2, ..., SA_{n-1}, and *PA* concurrently execute subtasks ST_1, ST_2, ..., ST_{n-1}, and ST_n divided from DT, the results from different mobile agents need to be combined into a final result at a network node Site on which agents agree. Results of subtasks ST_1, ST_2, ST_{n-1}, and ST_n from different agents can be described by the following Atomic Record type of Active State Space.

SAR=<String Results, Integer ApplicationID, Integer TaskID>

If ST_1, ST_2, ..., ST_{n-1} and ST_n are the same, agents SA_1, SA_2, ..., SA_{n-1}, and *PA* will compete for being alive at the same network node Site. The agent, which arrives at *Site* first, is regarded as the first one to accomplish this special subtask. This agent will be continuously alive, and others will exit their execution. It can be identified by the following Atomic Record as the Label in Resource Space of ASS whether an agent just transporting to *Site* is the first one to accomplish the special task.

LAR=<String State, Integer ApplicationID, Integer TaskID>

If a mobile agent cannot find the LAR in the current ASS, it is the first one to accomplish the special task identified by ApplicationID and TaskID, then it will put a LAR into the ASS. If a mobile agent finds the LAR in ASS, it is not the first one to accomplish the special task and will exit its execution.

If ST_1, ST_2, ..., ST_{n-1} and ST_n are different, all the results of subtasks ST_1, ST_2, ..., ST_{n-1}, and ST_n from different agents SA_1, SA_2, ..., SA_{n-1}, and *PA* need to be combined into a final result. The parent agent *PA* is responsible for the solution synthesis. The duplicated agents SA_1, SA_2, ..., and SA_{n-1} will exit their executions after putting their results into the current Resource Space of ASS at *Site*. The *inter-agent* interactions of the above scenario are graphically illustrated in Figure 2.

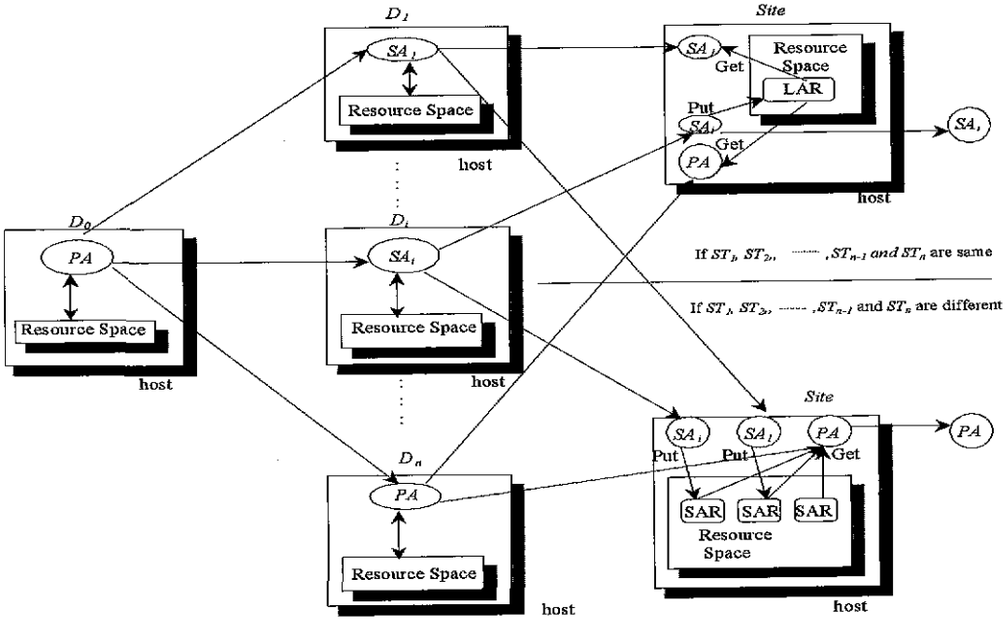

Figure 2: Inter-agent interactions by Resource Space of ASS

3.3 Working mode of workflow for mobile agents in MAT

An autonomous mobile agent should be able to exhibit goal-directed behaviours, which depends on a dynamically planned, continuous and autonomous workflow. In order to provide mobile agents with an autonomous workflow, we must solve two problems:

1. Maintaining the persistence of an agent's thread. This means the whole mapping (including the execution point of the thread) of an agent object must be restored after the agent migration. The weak mobility model, which is widely used in many mobile agent systems, such as Odyssey (Magic, 1998), Voyager (Object Space), Java-To-Go (Li and Messerschmitt, 1996), Aglets (Lange and Oshmina, 1998), Facile (Thomsen, Leth and Prasad, 1993), Tocoma (Johansen, Reuese and Schneider, 1995), Mole (Object Space) and Grasshopper (IVK, 1999), is not suitable for supporting the persistence of agents' threads. An agent's execution can only start from the beginning or from a special method rather than the stopped point after agent migration, so the persistence of an agent's thread cannot be maintained by weak mobility models. On the other hand, the strong mobility supports the persistence of agents' threads by transparently capturing the mapping of an agent's thread when an agent wants to migrate, transporting the captured mapping and restoring the transported mapping after the agent migration. But the strong mobility is often constructed on special languages. Systems based on special or specification-modified languages, such as Telescript (White, 1997), Agent Tcl (Gray, 1996), Ara (Peine and Stolpmann, 1997) and Sumatra (Acharya, Ranganathan and Saltz, 1996), are accepted by multiple platforms with difficulty. An evident example is that General Magic rewrites its mobile agent system, Telescript, into Odyssey by using Java in order to be widely accepted by heterogeneous platforms.
2. Permitting the modification of its behaviours when a mobile agent transports under the computational networks. This means a mobile agent cannot only execute a predefined workflow, but also modify or reconstruct its workflow in order to deal with the heterogeneity and dynamism of networks. Behaviours of an agent are exhibited by its code. Dynamic reconstruction of a workflow means that the agent kernel and its behaviours, which are defined by a user's objectives, should be loosely coupled and cannot be programmed in the same program unit because an agent needs to modify its code for adapting to dynamic networks.

Limitations of current technologies of agent migration necessitate the design of a new agent system model, MAT, for supporting autonomous mobile agents. In MAT, we program mobile agents with Mobile Thread Programming Model (MTPM). Contrary to many strong mobility technologies, MTPM is an application-level rather than a system-level model, that simulates the persistence of threads after agent migrations. An advantage of MTPM is that it is highly efficient. MTPM can simulate the persistence of an agent's thread without capturing, transporting and restoring the thread stacks and heaps with huge system-dependent information. MTPM does not need any modification to JVM specifications, and it uses two new mechanisms, Serialisation and Remote Method Invocation (RMI), provided by Java 1.1 and later releases, to provide persistence to an agent's thread. MTPM is outlined in Subsection 4.2. When it is created, a mobile agent plans its Distributed Task Plan (DTP) that is the implementation model of MTPM. An execution of a DTP generates a continuous and autonomous workflow for a mobile agent. When executing a DTP, a mobile agent can transport in networks, retrieve resources at remote hosts and asynchronously interact with other agents. At any time when current DTP fails, a mobile agent can replan a new DTP according to dynamic networks for an alternative accomplishment of a user's objectives. A DTP is a reusable template that is a static description of a distributed task for the execution by a mobile agent. An

execution of a DTP is an autonomous workflow for the agent. Whenever a mobile agent needs to modify its behaviours for adoption of the dynamism of networks, it can reach the goal by replanning its DTP. DTP is outlined in Subsection 4.2 and detailed in (Li and Zhang, 1999). The work mode of mobile agents in MAT is graphically illustrated in Figure 3.

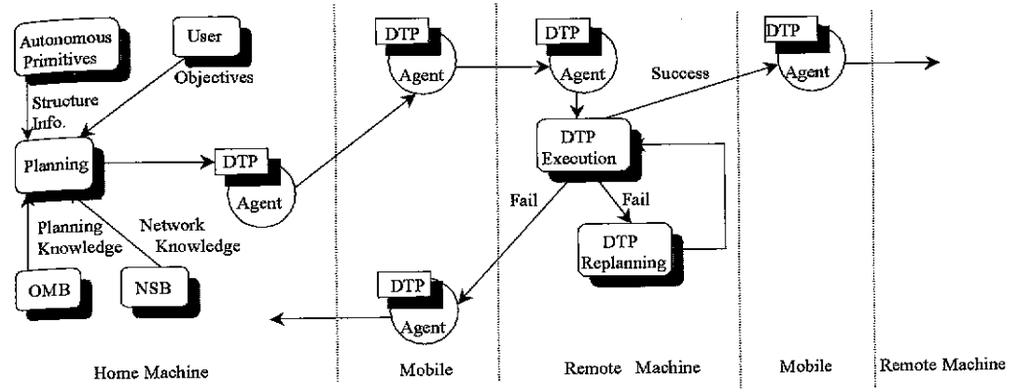

Figure 3: The planning, execution and replaning of DTP

4 IMPLEMENTING AUTONOMIES: MECHANISM OF MAT

4.1 ASS: distributed supporting environment

There are three motivations for the design of ASS. The first one is to provide communication facilities for autonomous mobile agents. Interactions among agents, by means of ASS, are not involved in any network communication and naming service. Interactions among agents are fully asynchronous, anonymous and localised. That means a mobile agent can put messages (called *Record* objects) in ASS, which another agent can get later. The second one is to provide a uniform description of heterogeneous resources that are accessed by mobile agents. ASS abstracts the resources of different systems into the data integrity so as to deal with heterogeneity and dynamism of the hosting execution environments. Mobile agents use virtual resources for accessing the real resources. ASS is responsible for mapping between abstract resources and real resources. The last motivation is to provide a secure interface between mobile agents and system resources. It is necessary for ASS to perform security checks on some sensitive operations such as deleting files because a malicious or bad-programmed agent possibly contains dangerous code.

4.1.1 Architecture of ASS

ASS is composed of *Resource Space*, *Security Space* and *Operation Space*. Virtual resources or messages for *inter-agent* communications are stored in the *Resource Space*. Any requirement to access *Resource Space* submitted to ASS by a mobile agent will fire an *Operation* object in the *Operation Space*. For the acceptance or rejection of a mobile agent's requirements, the *Operation* object needs to match the requirements with *Security Rules* in *Security Space*. This interaction procedure between mobile agents and hosting execution environments is called *Reflection of the Operation Object* in ASS. So every *Operation* object has a segment of *Reflection* codes. A successful reflection will change the states of ASS, and a failed reflection yields no influence on the ASS.

In an ASS, any component, such as a virtual resource, a security rule or an operation, can be installed and deleted dynamically by the ASS itself or mobile agents. The components, installed by

the ASS itself, are the common resources or protocols for all the agents and usually exist for a long period. The components, installed by agents, are the special resources or protocols for an application and usually exist for a short period. Components in the above three spaces are integrated into abstract description called an Active Record as defined below.

Definition 4: A *Record* in an ASS is composed of objects that are called *sub-records*, i.e. *<Record>::=<{subrec,}*>*, where subrec denotes any *sub-records* which compose the *Record*.

Definition 5: A *Record* is an *Atomic Record* **if** its every *sub-record is* not a *Record* object.

Definition 6: A *Record* is a *Coupled Record* if at least one of its *sub-records* is a *Record* object.

A file in a local file system can be structured with an *Atomic Record* object with the type of *<String path, String Type, Date UpdateTime, Username owner>*. Then the *Atomic Record* object of *<'/user/wli/document/', 'html', 2/22/99, wli@cowan.edu.au>* represents system file resources: all the *html* files that are located in the directory of *user/wli/document/*, owned by user *wli@cowan.edu.au* and updated in *February 22, 1999*.

A message, which a mobile agent puts in an ASS for communications with other agents, can be structured with an *Atomic Record* with the type of *<String Results, Integer ApplicationID, Integer TaskID>*. Every application has a unique identification *ApplicationID*, and every distributed task of the same application has a unique identification *TaskID*. Any agent can get the results of a distributed task by providing the *ApplicationID* and *TaskID* to the mechanism of the resource matching of the ASS.

A security rule of ASS can be structured with a *Coupled Record* object with the type of *<Integer AgentID, String Operation, AtomicRecord Rec>*. Then the *Coupled Record* object of *<*, 'r', Rec_1>* represents that any agent can read *Record* Rec_1. Another *Coupled Record* object of *<agentid, 'rdu', Rec_1>* represents that mobile agent identified by *agentid* can read, delete and update *Record* Rec_1.

Each Application ID, Task ID or Agent ID is 128 bits, and we use a special algorithm to generate IDs. Our ID algorithm is very similar to the algorithm of Microsoft for generating COM GUID (Microsoft). Microsoft demonstrated that the algorithm could generate different IDs in 3240 years at the generating speed of 1000000 IDs per second.

An *Operation* object of an ASS can be structured with an *Atomic Record* object with the type of *<String OperationName, Operation OpObj, Username Owner>*. Where the *OperationName* is the name of the *Operation*, *OpObj* is the instance of the *Operation* and the *Owner* is the entity (agent or local system) to install the *Operation*. Every *Operation* object has a *reflection* method. The *reflection* method is executed when the *Operation* object is called by ASS.

A retrieval requirement from a mobile agent to a *hosting execution environment* can be structured with a *Coupled Record* object with the type of *<Integer AgentID, String Operation, Boolean Synchronisation, AtomicRecord Rec>*. Then the *Coupled Record* object of *<agentid, 'r', true, <'/user/wli/document/', 'html', *, *>>* represents that a mobile agent, identified by *agentid*, wants to read all the *html* files in the directory of */user/wli/document/* in a *synchronous* mode. If the resource matching and the security inspection succeed, the ASS will return a vector of references to the required *html* files.

4.1.2 The reflection mechanism of ASS

ASS has three Object Spaces, which are *Resource Space*, *Security Space*, and *Operation Space*. The *Resource Space* stores *Atomic Records*. These records abstractly describe system resources of the *hosting execution environments* or messages that mobile agents put for asynchronous interactions.

The *Security Space* stores *Coupled Records*. These records describe security rules that determine which agents can do what kinds of operations on which records in the *Resource Space*. The *Operation Space* stores *Atomic Records*. These records describe *Operation* objects that are identified by names and are called by the ASS when mobile agents interact with the ASS. *Operation* objects can be installed by an ASS system or by mobile agents for some special reflection operations to interaction events.

Every *Agent Server* hosts an ASS object. When a mobile agent transports into a network node, the agent gets the reference to an ASS object from the *Agent Server*. An important feature of an ASS is that *Operation Records*, which are responding entities to interaction events, can be programmed, and can be installed dynamically by agents or an ASS itself. The whole procedure of an interaction event from the submission of an interaction requirement to the result return is graphically illustrated in Figure 4.

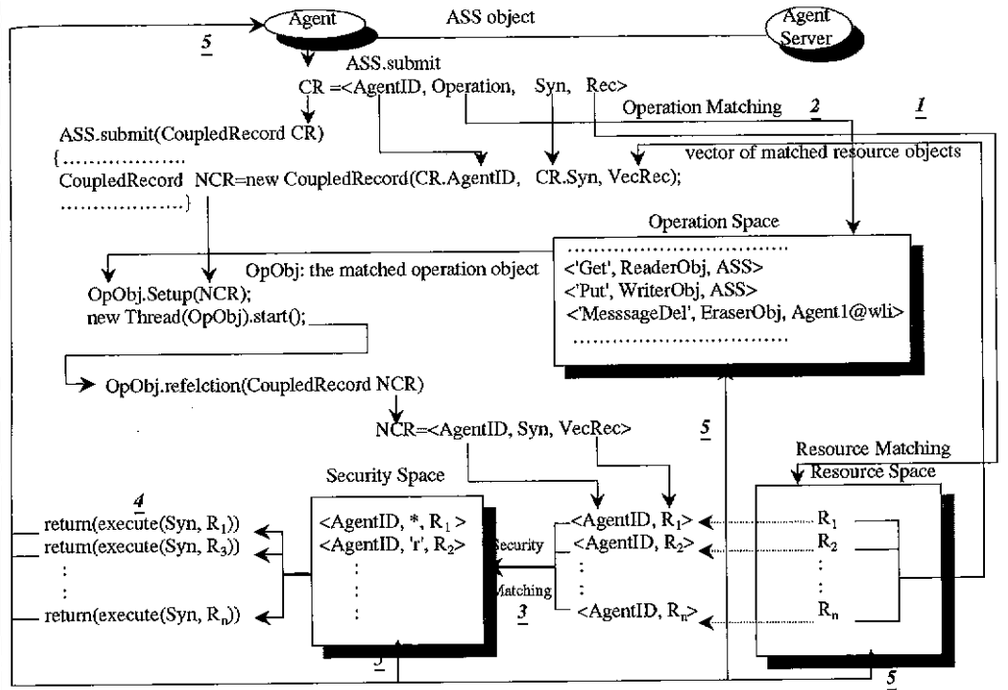

Figure 4: The reflection procedure of ASS object to an interaction requirement from a mobile agent

Every interaction requirement can be represented by a *Coupled Record*: *<AgentID, Operation, Syn, Rec>*, which means: a mobile agent identified by *AgentID* hopes to execute an *Operation* on all the *Records* matching the template record *Rec* in a synchronous mode *Syn*. The agent provides the above requirement to an *ASS* object with the *submit* method of the *ASS* object to which the agent has a reference. The reflection procedure of an ASS to the interaction requirement goes through the following steps:

1. The ASS object searches the whole *Resource Space* to identify records that match *Rec*, and uses these records to construct a vector *VecRec*.
2. The ASS object searches the whole *Operation Space* to identify a record that matches

Operation; generating a thread for the matched operation object *OpObj*, and starting the execution of the thread.

3. The *Operation* thread constructs a record *SR=<AgentID, R>* for each record *R* in vector *VecRec* and searches *Security Space* to identify the presence of a record that matches *SR*.
4. If a matched record is found in *Security Space*, *OpObj* will execute the required *Operation* on *R* in the synchronous mode *Syn*.
5. The results of the *Operation* can be returned to the agent or stored into *Resource Space*, *Security Space* or *Operation Space*.

Some ideas for the *ASS* design come from Jada (Ciancarini and Rossi, 1997). In Jada's space, objects are passive. *ASS* takes Jada's concepts one step further by allowing components of ASS to actively respond to interaction events and by allowing components to be installed, updated and deleted dynamically by the *ASS* itself or mobile agents.

4.2 Workflow template DTP for generating a continuous and autonomous workflow

In this subsection, we introduce the Distributed Task Plan (DTP), which is a flexible implementation model of MTPM for generating continuous and autonomous workflows for mobile agents. MTPM simulates the state persistence of an agent's thread by Serialisation and RMI without introducing any new spatial and time complexity. DTP is a workflow template that uses the programming paradigm of MTPM (Zhang and Li, 1999).

In order to obtain desirable autonomies for mobile agents, we firstly provide enough programming components and autonomous primitives, which are used to construct a DTP and further to provide mobile agents with autonomies in the navigation and the computation. In MAT, four kinds of autonomous primitives are defined for DTP designing.

1. Mobile primitives define the mobility of an agent. A mobile agent can merely transport itself to the next destination from the current network node by calling the *simple migration primitive*, or clone and transport each of its duplicates to different destinations by calling the *multiple migration primitive*.
2. Computational primitives define invocations of computational resources. A computational primitive specifies where to find the current computational procedure, how to load it and how to execute it. By using the computational primitives, a mobile agent realises that (a) the current computational procedure is carried by the agent or is resident at the visiting node; (b) the procedure should be started in a different process or loaded into its own process; and (c) how to run the computational procedure, e.g. synchronously or asynchronously.
3. Solution synthesis primitives define the combination of multiple solutions from different mobile agents. The solution synthesis is needed when a task is divided into several subtasks and executed by different mobile agents concurrently. It is highly efficient to divide a task into several subtasks and assign these subtasks to different mobile agents for execution when the task can be executed concurrently and multiple resources are available.
4. Control primitives define the execution flow of mobile primitives, computational primitives and solution synthesis primitives. Enough control structures of control primitives are needed to efficiently coordinate the executions of all the above three primitives.

Having defined primitives, we provide a reasonable model: DTP, which is used to depict distributed tasks for mobile agents by advantages of those pre-defined autonomous primitives. Normally, DTP is composed of all the four kinds of autonomous primitives.

A DTP consists of primitives, which are arranged into two lists. A list, called a control queue (CQ), only contains control primitives, and the other list, called a reusable primitive list (RPL),

contains any primitives except control primitives. The architecture of DTP is graphically illustrated in Figure 5 (concrete meanings of primitives of Figure 5 are defined in (Li and Zhang, 1999)).

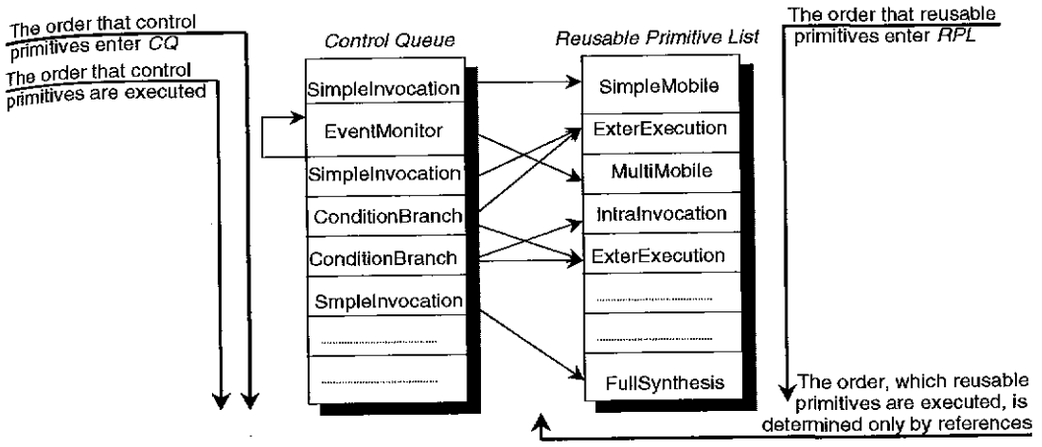

Figure 5: The architecture of DTP

When being created, a mobile agent plans its own DTP for the execution of a distributed task satisfying user's requirements. The planning procedure includes Objective Matching, Primitive Selection and DTP Generation by using users' requirements, network state information and task features. A mobile agent can also replan its DTP when the current DTP fails during execution. The planning of DTP is detailed in the next subsection.

A mobile agent has a reference to its DTP, and a DTP has a reference to a CQ. All the objects, from the agent itself, DTP to autonomous primitives, have a run() method. Every objects' run() method just calls the run() method of another object to which the former has a reference. The autonomous workflow of a mobile agent is generated when the mobile agent executes its DTP by running its run() method as shown in Figure 6.

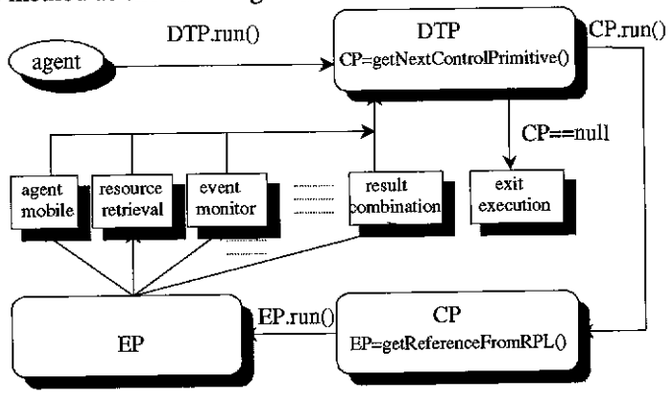

Figure 6: The continuous and autonomous workflow of agent by DTP

The order of primitives in a CQ is important. The control primitives in a CQ are executed sequentially. A control primitive in a CQ has one or more references to primitives in a RPL corresponding to which type the control primitive is. A reference to a control primitive in a CQ depicts a possible invocation to a primitive in a RPL. The order of primitives in a RPL is not

important because the invocations to them are determined only by references of control primitives in a CQ (see Figure 5). A RPL is just a repository of autonomous primitives that a mobile agent may need to execute when transporting in the underlying networks. So a mobile agent only executes control primitives in a CQ one by one, then further executes primitives in a RPL. An execution of a DTP is a continuous and autonomous workflow of a mobile agent. Constructing a complex workflow by using DTP provides a mobile agent with autonomy, flexibility and reusability in distributed applications.

4.3 Planning and replaning of DTP

According to a user's objectives, a DTP object should be planned either by a mobile agent or a static agent. In order to make mobile agents lightweight in code load and to support complex planning strategies, we move the planning algorithm of DTP (Li and Zhang, 1999) from the mobile agent in a previous version of MAT into the static agent (called *Planner*) into our current version. When a mobile agent is created, it gets a task from its owner. The task is described by the following *Atomic Record.*

OAR=<String TaskLable, String Objectives, Integer ApplicationID, Integer TaskID>

With following codes, the agent puts its *OAR* into a current ASS, and later gets its DTP object from the current ASS. The received DTP object is matched to the user's objectives described by *OAR*. Once a mobile agent gets its DTP object, the agent executes it.

```
ASS.submit(new CoupledRecord(this.AgentID, 'Put', true, OAR));
AtomicRecord  DTPID=new(*, OAR.Objectives, OAR.ApplicationID, OAR.TsakID);
AtomicRecord  DTPAR;
while ((DTPAR= ASS.submit(new CoupledRecord(this.AgentID, 'Get', true, DTPID)))==null)
     { Thread.sleep(time); };
DTP  DTPObject=DTPAR.DTPObject;
DTPObject.run();
```

When the agent fails to execute its current DTP object due to the dynamism of networks, it also uses the above codes to get a new DTP object, which is replanned by a Planner at its visiting network nodes, from the current ASS. The new DTP is an alternative workflow for the mobile agent to accomplish its owner's objectives in dynamic networks.

There is one Planner running on every distributed ASS. A Planner continuously gets users' objectives (described by *OAR*) from the current ASS, generates DTP objects and also puts them into the ASS using the following codes.

```
AtomicRecord  OARTemplate=new('task', *, *, *);
 while (true) {
   AtomicRecord  OARObject=ASS.submit(new CoupledRecord(this.AgentID, 'Get', true, OARTemplate));
   if (OARObject!=null) {
     DTP  DTPObject=planning(OARObject.Objectives);
     AtomicRecord  DTPAR=new AtomicRecord(DTPObject,OAR.Objectives, OAR.ApplicationID, OAR.TaskID);
     ASS.submit(new CoupledRecord(this.AgentID, 'Put', true, DTPAR)); }
else Thread.sleep(time); }
```


The procedure of DTP planning, running and replanning is graphically illustrated in Fig. 7.

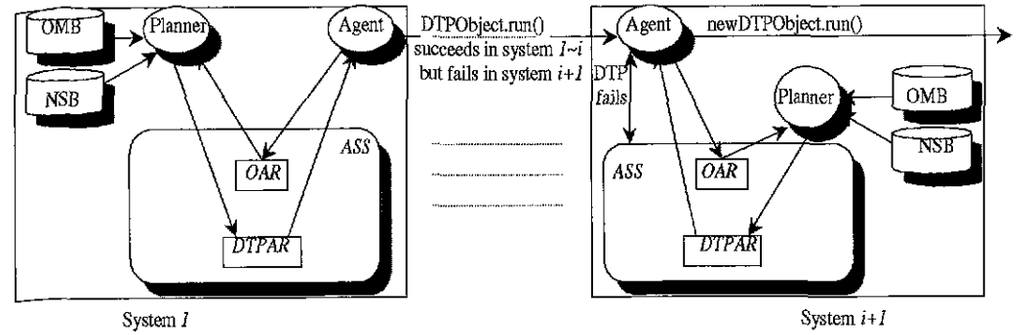

Figure 7: Planning, running and replanning of DTP

The Planner matches a user's objectives with entries in Objective Matching Base (OMB) to decide which primitives are used. The selected primitives are arranged into a CQ or a RPL according to their types. The references of primitives in the CQ to primitives in the RPL are set up at the same time. The developers of MAT applications define the OMB. An OMB consists of reusable entries. The matching method uses the algorithm that we prefer to call *Requirement Propagation* to match a user's objectives with pre-defined entries, which determine DTP planning. The Planner also retrieves a Network State Base (NSB) for information about current networks. A NSB is periodically updated by a MAT information server in order to reflect the dynamic networks. A concrete example of DTP planning is graphically illustrated in Figure 8. In this example a user wants to get a file *ProgramTable* from remote network node *butterfly* when the file is updated after *February 22, 1999.*

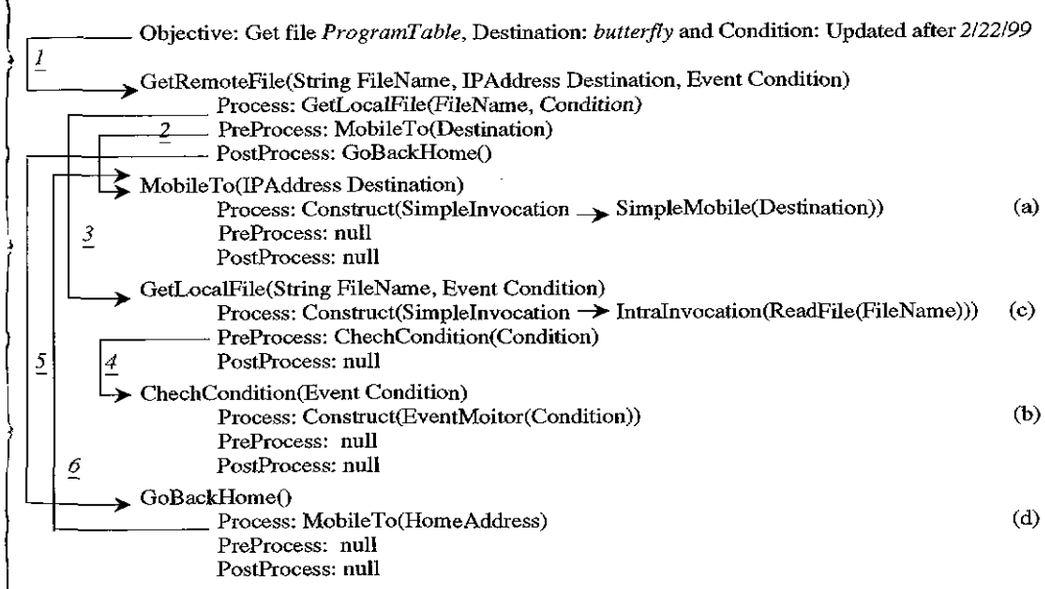

Figure 8: DTP planing: objectives matching with reusable entries in OMB

Each entry of OMB has four components, an entry name with parameters, *Process*, *PreProcess* and *PostProcess*. Process is executed after the execution of *PreProcess* and before the execution of *PostProcess*. The executions of *Process*, *PreProcess* or *PostProcess* can possibly access other entries, so the objective matching is the Requirement Propagation by reusing entries in OMB. In the above example, the objective matching is in the order of 1-2-3-4-5-6 and the DTP planning is in the order of (a)-(b)-(c)-(d). The workflow generated by executing the planned DTP will be:

Step 1: Transporting to *butterfly*,
Step 2: Monitoring file *ProgramTable* until it is updated after *February 22, 1999*,
Step 3: Reading the file, and
Step 4: Taking the file back the home machine.

The replanning of DTP is the same as the planing of DTP but the Planner uses the OMB and the NSB in the current node rather than those in the home machine of a mobile agent. The replanning of DTP gives a mobile agent an alternative workflow to accomplish its owner's objectives when current DTP fails.

5 CONCLUSION

Partially connected computers, such as laptops, personal digital assistants or modem-connected home computers, are involved with the development of the network, especially the wireless network. Applications launched from them are usually called mobile computation, which depends on the autonomies of mobile agents. MAT is a mobile agent system, which is embedded with high autonomies. As we know, many mobile agent systems, such as Telescript, Voyager and Aglets, do not or do not highly support autonomy of agents because their goal is not to support the novel Web application of mobile computation. Only the mobile agent system Agent Tcl pays some attentions to mobile computation. But none of them proposed the autonomy theory for the mobile agents. However, mobile computation will become a popular Web application with the development of wireless networks and more and more mobile devices, such as laptop computers and mobile phones. The feature, which distinguishes MAT from other agent systems, is that MAT concentrates on autonomies in the context of mobile agents. For supporting both mobility and autonomy, many concepts, paradigms and models, have been proposed for MAT implementation, which mainly include the autonomies of mobile agents, MTPM, DTP and ASS. All the proposed models are implemented currently. In MAT, a mobile agent is an autonomous entity that can plan its workflow and roam in networks for resource accessing or cooperation with other agents in asynchronous, anonymous and localized modes. By using MAT, a developer can use all kinds of templates, including DTP, Active Record and Object Matching Rule, to construct Web applications. A user of MAT applications only needs to provide his objectives and other necessary information such as features of tasks (concurrent or serial). Mobile agents can determine goal-directed work and cooperation modes according to users' requirements, and try alternatives when an effort fails.

MAT is a new prototype system implemented in Java that concentrates on providing agents with both mobility and autonomy. This paper summed up its current methodologies and implementation. A web application titled Intelligent Retrieval for the Information on the Web was constructed on MAT. This application supports the use of mobile devices. Another application titled Distributed Swarm is under construction. This application enables the Swarm (Swarm Design Group) into a distributed system. Our future work will focus on investigating the suitability and the effectiveness of MAT in Web applications, such as Internet information retrieval, electronic commerce and Computer Support Cooperative Work (CSCW). From feedback of the investigations, we may find problems in MAT, and make improvement to the methodology, model and their implementations.

ACKNOWLEDGMENTS

Authors would like to thank Mrs Anne Fuller for her proof reading of the paper. Authors cheerfully acknowledge the anonymous reviewers for their criticisms, comments and suggestions, which have been very helpful in improving the quality of this paper.

REFERENCES

ACHARYA, A., RANGANATHAN, M., and SALTZ, J. (1996): Sumatra: A language for resource-aware mobile programs. *Mobile Object System: Towards the Programmable Internet, Lecture Notes in Computer Science*, 1222:111-130, Berlin, Springer-Verlag.

BIC, F., FUKUDA, M. and DILLENCOURT, M. B. (1996): Distributed computing using autonomous objects. *IEEE Computer.*

BAUMANN, J., HOHL, F. and RADOUNIKLIS, N. (1997): Communication concepts for mobile agents. *Proc. of the First International Workshop on Mobile Agents, Lecture Notes in Computer Science*, 1219:123-135, Berlin, Springer-Verlag.

CAI, T, GLOOR, P. and NOG, S. (1996): Dartflow: A workflow management system on the web using transportable agents. *Technical Report TR96-283*, Department of Computer Science, Dartmouth College, Hanover.

CIANCARINI, P. and ROSSI, V. (1997): Jada: Coordination and communication for Java agents. *Mobile Object System, Lecture Notes in Computer Science*, 1222:213-226, Berlin, Springer-Verlag.

CIANCARINI, P., TOLKSDORF, R., and VITALI, F. (1998): Coordinating multiagent applications on the WWW: A reference architecture. *IEEE Transactions on Software Engineering*, 24(5):362-375.

MAGIC G. (1998): Introduction to the Odyssey API. Available at http://www.generalmagic.com/agents/ odysseyIntro.pdf.

GRAY G. (1996): Agent Tcl: A flexible and secure mobile-agent system. *Proc. of Fourth Annual Tcl/Tk Workshop*, Monterey, California.

GRAY, G., KOTZ, D., NOG, S., RUS, D. and CYBENKO, G. (1997): Mobile agents for mobile computing. *Proc. of the Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis*, Fukushima, Japan.

IKV (1999): Grasshopper. Available at http://www.ikv.de/products/grasshopper.html.

JENNINGS, N. J., SYCARA, K. and WOOLDRIDGE, M. (1998): A Roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7-38, Kluwer Academic Publishers.

JOHANSEN, D., RENESSE, R. and SCHNEIDER, F. B. (1995): An introduction to the TACOMA distributed system. Computer Science Technical Report 95-23, University of Tromso, Norway.

KOTZ, D., GRAY, R., NOG, S., RUS, D., CHAWLA, S., and CYBENKO, G. (1997): Agent TCL: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58-67.

LANGE, D. B. and OSHIMA, M. (1998): Programming and developing Java mobile agents with Aglets. Forthcoming Booking, Addson-Wesley.

LI, W. and ZHANG, M. (1999): Distributed task plan: A model for designing autonomous mobile agents. *Proc. of International Conference on Artificial Intelligence*, Las Vegas, 336-342.

LI, W., and MESSERSCHMITT, D. G. (1996): Java-To-Go. Technical report, Dept. of EECS, University of California, Berkeley, available http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/.

Microsoft, COM Specification. Available at http://www.microsoft.com/com/resources/comdocs.asp

Object Space, Voyager Core Technology 2.0 User Guide. Available at http://www.objectspace.com/ developers/voyager/white/voyager20.pdf.

PEINE, H., and STOLPMANN, T. (1997): The architecture of the Ara platform for mobile agents. *Proc. of the First International Workshop on Mobile Agents, Lecture Notes in Computer Science*, 1219:50-61, Berlin, Springer-Verlag.

RUS, D., GRAY, R., S., and KOTZ, D. (1997): Transportable information agents. *Proc. of International Conference on Autonomous Agents*, 228-236.

Swarm Design Group, A Tutorial Introduction to Swarm. Available at http://www.swarm.org/csss-tutorial/frames.html.

STRAßER M., BAUMANN, J., and HOHL, F. (1997): Mole - A Java based mobile agent system. *Special Issue in Object Oriented Programming*, M. Mühlhäuser (ed.), Dpunkt Verlag, 301-308.

THOMSEN, M., LETH, L., and PRASAD, S. (1993): Facile Antigua release programming guide. *Technical Report ECRC-93-20*, European Computer Industry Research Centre, Munich, Germany.

WHITE, J. E. (1997): Mobile agents. *Software Agents*, MIT Press.

ZHANG, M., and LI, W. (1999): Persisting autonomous workflow for mobile agents using a mobile thread programming model. *Approaches to Intelligent Agents, Lecture Notes in Artificial Intelligence*, 1733:84-95, Berlin, Springer-Verlag.

BIOGRAPHICAL NOTES:

Wei Li is Associate Professor in Computer Science, in the Department of Information Management, Capital University of Economics and Business, Beijing P.R. China. Currently he is a Post Doctoral Research Fellow in the Faculty of Informatics and Communication, Central Queensland University, Australia. He was awarded his Ph.D. degree in Computer Science in the Institute of Computing Technology (ICT), Chinese Academy of Sciences, in July of 1998. His major is in the area of Artificial Intelligence and Distributed Computation.

Minjie Zhang is a Senior Lecturer in the School of Information Technology and Computer Science, University of Wollongong, Australia. She received her Bachelor of Computer Science degree from Fudan University, China in 1982 and received her Ph.D. degree in Computer Science from the University of New England, Australia in 1996. She is a member of the IEEE, IEEE Computer Society, and ISCA. She is the author, or co-author, of over thirty research papers. Her research interests include distributed artificial intelligence, intelligent information retrieval, and software engineering.